W8.1 Continuing Classes

• friend Functions and friend Classes
• Using the this Pointer
• Cascading Function Calls

7.4 friend Functions and friend Classes

• friend function and friend classes
  – Can access private and protected members of another class
  – friend functions are not member functions of class
    • Defined outside of class scope
• Properties of friendship
  – Friendship is granted, not taken
  – Not symmetric (if A is friend of B, B is not necessarily a friend of A)
  – Not transitive (if A is friend of B, B is friend of C, A is not necessarily a friend of C)

When to Use a friend

• Using friend functions can enhance performance.
• friend functions may be used to overload operators for classes and to create iterator classes.
  – Objects of iterator class used to successively select items or perform an operation on items in a container class (Ch. 7.9)
  – Objects of container classes are capable of storing items.
• Using friend functions is appropriate when member function cannot be used (operator overloading, see Ch. 8.4, later)

7.4 friend Functions and friend Classes

• friend declarations
  – To declare a friend function
    • Type friend before the function prototype in the class that is giving friendship
      friend int myFunction( int x ); should appear in the class giving friendship
  – To declare a friend class
    – Type friend class Classname in the class that is giving friendship
    – if ClassOne is granting friendship to ClassTwo,
      friend class ClassTwo;
      should appear in ClassOne’s definition

Outline

1. Class definition
1.1 Declare a class
1.2 Function definition
1.3 Initialize a class object

Program Output

31    cout << "counter.x after call to setX friend function: ";
32    setX( counter, 8 );  // set x with a friend
33    counter.print();
34    return 0;

changing private variables allowed.

private data was changed.
void cannotSetX( Count &c, int val )
// but cannot because it is not a friend of Count.
// Function tries to modify private data of Count,
// which is not accessible because it is private.

private:
    void print() const { cout << x << endl; }  // output
    Count() { x = 0; }                   // constructor

public:
    class Count {
    using std::endl;
    using std::cout;
    
    #include <iostream>

    // using the this pointer to refer to object members.
    
    // Example of cascaded member function calls
    
    // Member functions setHour, setMinute, and setSecond
    // all return *this (reference to an object)
    // For object t, consider
    // t.setHour(1).setMinute(2).setSecond(3);
    // Executes t.setHour(1), returns *this (reference to
    // object) and the expression becomes
    // t.setMinute(2),setSecond(3);
    // Executes t.setMinute(2), returns reference and
    // becomes t.setSecond(3);
    // Executes t.setSecond(3), returns reference and
    // becomes t;
    // Has no effect

    int main()
```cpp
int main() {
    Time t; // default constructor
    t.setTime( 20, 20, 20 ).printStandard();
    t.printStandard().setTime();
}
```

---

**Outline**

**Program Output**

```
New standard time: 8:20:20 PM
Standard time: 6:30:22 PM
Military time: 18:30
```

---

All three methods have the same result.

Notice the `return *this` function returns a reference to an object.

Specify object in function definition.

**Cascading function calls.**

Calling `setTime` after `printStandard` does not return a reference to an object.

`printStandard()` sets time, `setTime()` sets time.

Cascading function calls `printStandard` must be called after `setTime` because `printStandard` does not return a reference to an object.

`printStandard()`, `setTime()`, would cause an error.

```
return *this;   // enables cascading
```

**Notice the Time & function returns a reference to a Time object.**

Specify object in function definition.

---

**Outline**

**Program Output**

```
New standard time: 8:20:20 PM
Standard time: 6:30:22 PM
```

---

All three methods have the same result.

Notice the `return *this` function returns a reference to an object.

Specify object in function definition.

**Cascading function calls.**

Calling `setTime` after `printStandard` does not return a reference to an object.

`printStandard()` sets time, `setTime()` sets time.

Cascading function calls `printStandard` must be called after `setTime` because `printStandard` does not return a reference to an object.

`printStandard()`, `setTime()`, would cause an error.

```
return *this;   // enables cascading
```

**Notice the Time & function returns a reference to a Time object.**

Specify object in function definition.

---

**Outline**

**Program Output**

```
New standard time: 8:20:20 PM
Standard time: 6:30:22 PM
```

---

All three methods have the same result.

Notice the `return *this` function returns a reference to an object.

Specify object in function definition.

**Cascading function calls.**

Calling `setTime` after `printStandard` does not return a reference to an object.

`printStandard()` sets time, `setTime()` sets time.

Cascading function calls `printStandard` must be called after `setTime` because `printStandard` does not return a reference to an object.

`printStandard()`, `setTime()`, would cause an error.

```
return *this;   // enables cascading
```

**Notice the Time & function returns a reference to a Time object.**

Specify object in function definition.